Multithreaded Verification
by
Context Inference

Tom Henzinger  Ranjit Jhala  Rupak Majumdar
Multithreaded Programs

OS, WebServers, Databases, Embedded Systems

Curse of **Interleaving**
- Non-deterministic scheduling
- Exponentially many behaviors: hard to detect, reproduce errors

**Testing** exercises a fraction of possible behaviors
Safety Verification by State Exploration

Is there a **path** from **Initial** to **Error**?
Problem: State Explosion

Is there a path from Initial to Error?
**Problem: State Explosion**

1. **Data**
   - Infinitely many valuations for program variables

2. **Control**
   - $k$ threads, $m$ locations = $m^k$
   - $k=4, m=100$, states = 1 billion
   - Unbounded threads?
Problem: State Explosion

1. Data
   Infinitely many valuations for program variables

2. Control
   $k$ threads, $m$ locations = $m^k$
   - $k=4, m=100$, states = 1 billion
   Unbounded threads?
Solution: Abstract Irrelevant Detail

1. Data
Infinitely many valuations for program variables

Observation:
- Few relevant variables, relationships
- Track predicates (relationships) instead of values

1. Predicate Abstraction

2. Control
$k$ threads, $m$ locations = $m^k$
- $k=4, m=100$, states = 1 billion
Unbounded threads?

Observation:
- Analyze system as Thread + Context
- Context: Abstraction of other threads (w.r.t. property)

2. Context Abstraction
Plan

1. Abstractions against State Explosion
   • Data : Predicate Abstraction
   • Control : Context Abstraction

2. Thread-Context Reasoning

3. Context Inference

4. Experiments

5. Related Work
Example Property: Data Races

A data race on $x$ is a state where:

- **Two** threads can access $x$
- One of the accesses is a **write**

Unpredictable, undesirable
Example Program

Check for races on \( x \):

Initially: \( s \) is 0

1st thread into `atomic`:
- sets `old` to 0 (value of \( s \))
- sets \( s \) to 1
- passes test before access

Later threads:
- set `old` to 1
  (value set by 1st thread)
- fail test before access
  (until the 1st thread is done)
Plan

1. Abstractions against State Explosion
   - Data : Predicate Abstraction
   - Control : Context Abstraction

2. Thread-Context Reasoning

3. Context Inference

4. Experiments

5. Related Work
Predicate Abstraction

```plaintext
1: while(1){
   atomic{
   2: old := s;
   3: if(s==0){
      4: s := 1;
      // do_work()  M
   }
   
   5: if(old==0){
      6: x++;
      7: s:=0;
   }
   }
}
```

Predicates on Variables
s=0, old=0

Reachability Graph

Q: What about other threads?
Plan

1. Abstractions against State Explosion
   - Data: Predicate Abstraction
   - Control: Context Abstraction

2. Thread-Context Reasoning

3. Context Inference

4. Experiments

5. Related Work
Plan

1. Abstractions against State Explosion
   - Data: Predicate Abstraction
   - Control: Context Abstraction

2. Thread-Context Reasoning

3. Context Inference

4. Experiments

5. Related Work
Threads and Contexts

Assume threads run same code

**Context:**
Summary of all other threads
- Precise enough to check property

**System = Thread + Context**

Q: What about other threads?

Q: What does a Context look like?
Contexts = Thread Summary + Counting

**Context:**
- Abstraction of all other threads

1. **Summarize** a single thread
2. Multiple threads by **counting**
Multiple Threads by Counting

Initial loc = 1, other = 0

Operations
1. Pick edge w/ source counter > 0,
2. Source counter -1
   Target counter +1
   Havoc variables on edge,
   Assume predicate on target

Unbounded threads
k-Counter Abstraction:
   Value > k abstracted to 1
   for k=1, values: 0, 1, 1
Plan

1. Abstractions against State Explosion
   - Data: Predicate Abstraction
   - Control: Context Abstraction = Summary + Counting

2. Thread-Context Reasoning

3. Context Inference

4. Experiments

5. Related Work
Plan

1. Abstractions against State Explosion
   - Data: Predicate Abstraction
   - Control: Context Abstraction = Summary + Counting

2. Thread-Context Reasoning

3. Context Inference

4. Experiments

5. Related Work
Thread-Context Reasoning

1. **Data:** Predicates
   - Use Context
   - Build finite Reach Graph
   - Check Error unreachable

2. **Control:** Summary, $k$
   - Verify Context Sound
   - Check Summary Overapproximates single Thread’s behavior

**Shared Memory**

Reach Graph

Computed Summary

Given Summary
Thread-Context Reasoning

1. **Data:** Predicates

2. **Control:**
   - Summary, $k$
   - Thread-Context Reasoning

Given Summary

Computed Summary

Assume-Guarantee

(Use)       (Verify)

Safe
Thread-Context Reasoning

Given an Abstraction:
1. Data: *Predicates*  2. Control: *Summary, k*

**Q:** How to find *predicates, summary, k*?
Plan

1. Abstractions against State Explosion
   - Data: Predicate Abstraction
   - Control: Context Abstraction = Summary + Counting

2. Thread-Context Reasoning

3. Context Inference

4. Experiments

5. Related Work
Inference: Build Summary

Reach Graph

Error

Trace

Abstraction

Preds: \( P_0 \)  
Ctr: \( k_0 \)

Summarize

Safe
Inference: Trace Analysis

**Refine** using Trace

Either:
1. Add **new predicates**
2. Increase **k**
Inference: Build Summary

Abstraction
Preds: $P_1$, Ctr: $k_1$
Context Inferred

1. Reach Graph
2. Summarize

Assume-Guarantee

Shared Memory

Safe
Context Inference

1. Init. Abstraction
   Preds: \( P_0 \)  Ctr: \( k_0 \)

2. Init. Summary
   Summary: ;

3. Refine using Trace

4. Reach Graph
   Safe?
   Feasible?
   NO (trace)

5. Update Summary
   NO
   YES
   μ?

6. Output
   Safe
   Error

Trace Analysis

Build Summary
Plan

1. Abstractions against State Explosion
   - Data: Predicate Abstraction
   - Control: Context Abstraction = Summary + Counting

2. Thread-Context Reasoning

3. Context Inference
   - Example

4. Experiments

5. Related Work
Ex: Races on $x$

```c
while(1) {
    atomic {
      old := s;
      if(s==0) {
        s := 1;
      }
    }
    // do_work()
    M
    if(old==0) {
      x++;
    }
    s := 0;
}
```

Abstraction

$Preds = ;$

$k=1$

Build Summary

Reach Graph

Summarize

Control-Flow Graph
Ex: Races on \( x \)

```c
1: while(1){
   atomic{
   2:   old := s;
   3:   if(s==0){
   4:       s := 1;
   5:   }
   6:   }
   // do_work()
   M
   7:   if(old==0){
   8:       x++;
   9:       s:=0;}
   }  
```

Abstraction

*Preds =;  K=1*
**Ex: Races on X**

Thread 1

```c
while(1)
{
    atomic{
        old := s;
        if(s==0)
        {
            s := 1;
        }
    }
    // do_work()
}
```

```c
if(old==0)
{
    x++;
    s:=0;
}
```

---

Trace Analysis

**Trace**

Thread 0

```c
assume (True)
old := s
assume (s==0)
s:= 1
// do_work()
assume (old==0)
//write x enabled
```

```c
if(old==0)
{
    x++;
    s:=0;
}
```

---

**Abstraction**

Preds =

k=1

---

x++
Thread 0
\begin{align*}
\text{assume} & \quad (\text{True}) \\
\text{old} & \quad := \quad s \\
\text{assume} & \quad (s==0) \\
\textbf{old} & \quad := \quad s \\
\textbf{s} & \quad := \quad 1 \\
\text{// do\_work()} \\
\text{assume} & \quad (\text{old}==0) \\
\text{//write x enabled}
\end{align*}

Thread 1
\begin{align*}
\text{assume} & \quad \text{(True)} \\
\textold & \quad := \quad s \\
\text{assume} & \quad (s==0) \\
\textbf{s} & \quad := \quad 1 \\
\text{// do\_work()} \\
\text{assume} & \quad (\text{old}==0) \\
\text{//write x enabled}
\end{align*}

\textbf{s} \quad \text{is set to} \quad 1

\textbf{Infeasible branch}

\textbf{New Predicate}
\textbf{s} = 0
Ex: Races on $X$

Abstraction

**Preds:** $s=0$, $old=0$

$k=1$

Build Summary

Local Pred. $old=0$
- Prunes infeasible paths
- Invisible to other threads
- Quantified away
Ex: Races on X

Abstraction
Preds: s=0, old=0
k=1

Build Summary

Context changes s

1: while(1){
   atomic{
      2: old := s;
      3: if(s==0){
         4: s := 1;
         // do_work()
         5: if(old==0){
            6: x++;
            7: s:=0;
         }
      }
   }
}

Regional Pred. old=0
- Cuts locally infeasible paths
- Invisible to other threads
- Quantified away
Ex: Races on $x$

1: while(1) {
    atomic {
        2: old := s;
        3: if(s==0) {
            4: s := 1;
        }
    }
    // do_work()
}

5: if(old==0) {
    6: x++;
    7: s := 0;
}

SAFE
No Races on $x$

Abstraction
Preds: $s=0$, $old=0$
k=1
Plan

1. Abstractions against State Explosion
   - Data : Predicate Abstraction
   - Control : Context Abstraction = Summary + Counting

2. Thread-Context Reasoning

3. Context Inference
   - Example

4. Experiments: Races in NesC Programs

5. Related Work
Race Checking in NesC Programs

PL for Networked Embedded Systems [Gay et al. 03]

- **Interrupts** fire events, which fire other events or post tasks which run asynchronously

- Race-freedom important
  - Flow-based analysis
  - Non-trivial synchronization idioms

- Compiled to C
Lock-based Synchronization

A data race on $x$ is a state where:

- **Two** threads can access $x$
- One of the accesses is a **write**

**Unpredictable, undesirable**

**Synchronization**: Must hold **lock** when accessing $x$
NesC Synchronization Idioms

```c
atomic{
    old := state;
    if(state==0){
        state := 1;
    }
}
M
if(old==0){
    x++;
    state := 0;
}
```

**State-based**
Case Study: *sense.nc*

```plaintext
atomic{
  old := state;
  if (state == 0) {
    state := 1;
    M
  }
}
M
if (old == 0) {
  x++;
  M
}
```

**Interrupt 1 fires**

- old := state
- if (state == 0) {
  state := 1
  M
- assume (old == 0) {
- about to write x

**Interrupt 1 handler disables interrupt 2**

**BLAST finds information proves no races**

**Interrupt 2 fires**

- state := 0

**Interrupt 1 fires**

- old := state
- assume (state == 0) {
  state := 1
  M
- If (old == 0) {
- about to write x
```
## NesC Race Checking Results

<table>
<thead>
<tr>
<th>Program (size)*</th>
<th>Variable</th>
<th>Preds</th>
<th>Summary</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SecureTosBase</td>
<td>gTxState</td>
<td>11</td>
<td>23</td>
<td>7m38s</td>
</tr>
<tr>
<td>(9539 lines)</td>
<td>gTxByteCnt</td>
<td>4</td>
<td>13</td>
<td>1m41s</td>
</tr>
<tr>
<td></td>
<td>gTxRunCrc</td>
<td>4</td>
<td>13</td>
<td>1m50s</td>
</tr>
<tr>
<td></td>
<td>gTxProto</td>
<td>0</td>
<td>9</td>
<td>12s</td>
</tr>
<tr>
<td></td>
<td>gRxHeadIdx</td>
<td>8</td>
<td>64</td>
<td>20m50s</td>
</tr>
<tr>
<td></td>
<td>gRxTailIdx</td>
<td>0</td>
<td>5</td>
<td>2s</td>
</tr>
<tr>
<td>Surge</td>
<td>rec_ptr</td>
<td>4</td>
<td>23</td>
<td>1m18s</td>
</tr>
<tr>
<td>(9637 lines)</td>
<td>gTxByteCnt</td>
<td>4</td>
<td>15</td>
<td>1m34s</td>
</tr>
<tr>
<td></td>
<td>gTxRunCrc</td>
<td>4</td>
<td>15</td>
<td>1m45s</td>
</tr>
<tr>
<td></td>
<td>gTxState</td>
<td>11</td>
<td>35</td>
<td>9m54s</td>
</tr>
<tr>
<td>Sense (3019 lines)</td>
<td>tosPort</td>
<td>6</td>
<td>26</td>
<td>16m25s</td>
</tr>
</tbody>
</table>

* Pre-processed
Limitations

- **Predicates**
  - How complex are the required invariants?
  - Are quantified invariants (predicates) needed?
  - Complex data structures

- **Concurrent Interaction Depth**
  - Iterations to fixpoint = Interaction depth ...

- **State Explosion**
  - Counter Abstraction blows up for large counter values $k > 3$ ...
  - Possible states = $P \mathcal{E} k^n$
Related Work: Locks

Dynamic LockSet
- [Dinning-Schonberg 90]
- [Savage et al. 97]
- [Cheng et al. 98]
- [Choi et al. 02]

Type-based
- [Flanagan-Freund 00]
- [Bacon et al. 00]
- [Boyapati et al. 02]

Static LockSet
- [Sterling 93], [Engler-Ashcraft 03]

Object Usage Graph
- [von Praun-Gross 03]

1. Infer some lock(s) that protect x
2. Check lock(s) held when accessing x
3. Report error if lock(s) not held

Scalable

Restricted to locking
Related Work: State Exploration

State Exploration (Model Checking)

[Lubachevsky 83]
[Godefroid 97]
[Holzmann][Havelund-Visser]
[Dwyer-Hatcliff][Avrunin-Clarke]
[Musuvathi-Dill-Engler 02]
[Yahav 01]

State Explosion

Any Sync. Idiom
Fixed #threads
Manual Abstraction
Related Work: Assume-Guarantee

- **Sequential Programs** (Procedure Pre/Postconditions)
  [Hoare 71]
- **Concurrent Systems**
  [Chandy-Misra 81]
  [Abadi-Lamport 93]
- **Hardware Verification**
  [Alur-Henzinger 96][McMillan 97]
  [Eiriksson, McMillan, Henzinger-Qadeer-Rajamani]
- **Software Verification**
  [Jones 83] [Flanagan-Qadeer-Freund-Seshia 02]
- **Automating via Machine Learning**
  [Giannapokopolou-Pasareanu-Barringer 03][Alur et.al. 05]
To sum up ...

- Multithreaded programs are hard to verify
  - Data Explosion from many possible values of variables
  - Control Explosion from thread interleaving

- Data Abstraction
  - Track important relationships via predicates

- Control Abstraction
  - Thread Summary = Abstract state machine
  - Context = Summaries + Counting

- Thread-Context Reasoning
  - System = Main Thread + Context

- Iterative Context Inference
Thank You!

http://www.eecs.berkeley.edu/~blast